VIDEO GAME DEVELOPMENT USING UNITY: FIRST PERSON SHOOTER

A Mini Project Report

submitted by

AMAN ABDUL MALIK K P(MES20MCA-2004)

to

the APJ Abdul Kalam Technological University
in partial fulfillment of the requirements for the award of the Degree

of

Master of Computer Applications

Department of Computer Applications

MES College of Engineering
Kuttippuram, Malappuram - 679 582

February 2022

DECLARATION

I undersigned hereby declare that the project report **VIDEO GAME DEVELOPMENT USING UNITY: FIRST PERSON SHOOTER**, submitted for partial fulfillment of the requirements for the award of degree of Master of Computer Applications of the APJ Abdul Kalam Technological University, Kerala, is a bona fide work done by me under supervision of Mr. Hyder Ali K, Associate Professor, Department of Computer Applications. This submission represents my ideas in my own words and where ideas or words of others have been included, I have adequately and accurately cited and referenced the original sources. I also declare that I have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

Place: Kuttippuram

Date: 04/03/2022

AMAN ABDUL MALIK K P(MES20MCA-2004)

DEPARTMENT OF COMPUTER APPLICATIONS
MES COLLEGE OF ENGINEERING, KUTTIPPURAM

CERTIFICATE

This is to certify that the report entitled **VIDEO GAME DEVELOPMENT USING UNITY: FIRST PERSON SHOOTER** is a bona fide record of the Mini Project work carried out by **AMAN ABDUL MALIK K P(MES20MCA-2004)** submitted to the APJ Abdul Kalam Technological University, in partial fulfillment of the requirements for the award of the Master of Computer Applications, under my guidance and supervision. This report in any form has not been submitted to any other University or Institution for any purpose.

Internal Supervisor(s) External Supervisor(s)

Head Of The Department

Acknowledgements

First of all, I would like to give thanks to God for giving me the opportunity to complete and submit this project. Without the help and blessings of God the Almighty, it would not have been possible to accomplish this project.

With great respect I express my sincere thanks to Dr.K.A Navas, Principal, MES College of Engineering, providing facilities for this project. I would also like to express my gratitude towards my supportive and encouraging project coordinator, Ms.Priya J.D Assistant professor, Department of Master of Computer Applications, for allowing me to do this project and for her inspiring guidance, reliability, constructive criticism and challenging but motivational feedback throughout the course of this project.

I would like to express my sincere thanks to Mr.Hyderali K, Associate professor, Head of the Department and my internal guide and all of the faculty members of the Department of Master of Computer Applications for equipping me with the skills and knowledge of our field of education required to accomplish this project. Your contribution will doubtlessly be acknowledged and I will always remain grateful to you all. Last but not least my graceful thanks to my parents, friends and also the persons who supported me directly and indirectly during the project.

AMAN ABDUL MALIK K P(MES20MCA-2004)

Abstract

This report focuses on the development of a 3D FPS (First Person Shooter) RPG (Role Playing Game) Desktop game, Captain-G. The project is based on Retro shooting games. This game contains a challenging single level for the the player to play. It has multiple achievable weapons and power ups, computer controlled intelligent enemies, both short and Ranged weapons to give the player a more interesting game play and other interesting features like loot items and weapon pickups.

This project explores a new dimension to the traditional RPG games by mixing the features of open world shooting games where player shoots enemies to survive and can navigate throughout the game world. With the simplicity of this game it simply aims to bring fun and make you look back to your childhood's Doom, Quake games with some new modern features.

Keywords: 3D, First-person shooter (FPS), Role-Playing Game (RPG), Desktop Game

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

The gaming industry has had much growth in recent years. It is a great industry to get involved in as it allows creativity, innovation and freedom for developers and hobbyists. They get a chance to experiment with all forms of media including sound design, environment design and programming. As there has been an explosion of the new 'Indie' market of games, the gaming industry now allows smaller-scale games to be developed and released more freely than in the past. Today it has become so easy to create a game. A large studio is no longer needed and the freely available tools make it so simple to create an idea from the comfort of your own home. Indie games show more innovation and developers are willing to take risks on their games. 'Indie' games also have a large market base with many of the games being released on PC, Android etc.

There is a massive popularity for not just first person shooter games, but particularly innovative survival shooter games. I put a lot of research into trying to find out what users want from a game. I had to decide between a first person shooter or a third person shooter, and what type of environment would be best to build my game. I finally decided to go with a FPS in a retro style environment. I decided to go with a 3D game over a 2D game as I think it makes the game that extra bit challenging and interesting.

1.1.1 Motivation

In the fast growing field of software engineering and development and even more rapidly growing sector of game development the future is hard to predict. In general software project is a project focusing on the creation of software. Consequently, Success can be measured by taking a look at the resulting software. In a game project, the product is a game. But and here comes the point: A game is much more than just its a software. It has to provide content to become enjoyable. Just like a web server: without content the server is useless, and the quality cannot be measured. This has an important effect on the game project as a whole. The software part of the project is not the only one, and it must be considered in connection to all other parts: The environment of the game, the story, characters, gameplays, the artwork, and so on.

Video games are not just any computer software which are made to benefit user's daily life, games are rather made for user's entertainment purpose, so more than anything we need to pay attention to what the user wants from the game, how to make it more entertaining, just making any game will not do, that is why it's more challenging because I always have to carefully consider if I'm developing it correctly to entertain users. I also have to invest a lot of time on the proper game designing to make it visually accepted. And to add that a game requires a lot of scripts. The scripts are like pieces of a puzzle which you need to put all of them together to make it work.

Despite the economic instability and crisis deeply affecting the world, the analysts published that the game industry has grown at a rate of 57% surprisingly. Even as I type these words millions of people are playing games in front of their computers.The reason of this growth can be stated that the game industry can appeal any user with different tastes. And there's no wonder that the gaming industry became the number one entertainment industry in the world. Thus I think a video game is a perfect project to prove myself as a computer application student and this will be a stepping stone to the world's number one entertainment industry.

1.2 Objective

The aim of this project was to create a fully functional FPS 3D game. The overall aim of my game was to create a tense and unnerving experience for the player. I chose a sci-fi environment setup as atmosphere is essential for the game. As I have path finding on my enemies, they will not attack until the player is in a certain range. They might not always be visible to the player as the scene has many hiding places. This adds a level of fear in the game as you will not know how close the next enemy is or when they will attack. I also included audio and sound effects to support the atmosphere of the game. The purpose of this project is to develop a first person shooter game with good graphics, audio and animations made with basic technologies and a low budget. I wanted to make a game that was easy to install and show what can be done with just a bit of time and effort. I have already inspired one of my friends to create a game as I had no experience prior and they were impressed with what I could accomplish in a short amount of time. The game is intended to be used on a PC running windows with a keyboard and a mouse.

I chose the Unity 3D game engine to develop my game as it has a large community which provides learning support from the Unity website and forums. There are many tutorials also available on YouTube. You can code your game in either C# or JavaScript which made it ideal, as both languages is very common and lot of learning materials can be found Online or libraries. The game will run on a Windows PC. This gives me the option of publishing my game to the popular PC online gaming market Steam. This would make my game available to many users online.

The game engine used is Unity3D. The game scripts is written mainly in C#. Visual Studio 2019, the main IDE which is used with Unity was used to edit scripts. Assets in my game were sourced from the Unity asset store.

1.3 Report Organization

This project report is divided into four sections. Section 2 describes literature survey. Section 3 describes the methodology used for implementing the project. Section 4 gives the results and discussions. Finally Section 5 gives the conclusion.

Chapter 2

Literature Survey

2.1 Unity Game Development Engine

2.1.1 Unity Game engine

In this section of the proposed paper, the literature survey has been conducted. The documentation of relevant works and examination of the collected sources fundamentally is demonstrated here. Therefore, the literature survey has been considered as one of the vital aspects of this report as it supports authors as well the readers in understanding and examining several facets that the relevant to the current topic and are conducted previously. Unity, a multi-platform game engine, commercially available and is used for 2d and 3D video games production accompanied by visualizations and non-game interactive simulations. Moreover, Unity is one of the popular engines for games that is easily accessible, and in the current epoch is communal amid the developers due to its ease, flexibility, efficiency, and power consumption. Multiple tools have been featured in the Unity Editor that permits rapid iteration and editing in the cycles of development comprised of smart previews play mode in real-time [1]. Furthermore, Unity is offered on Mac, Linux, and Windows it contains an artist friendly range of tools for immersive designing and game worlds, in addition to a strong developer tools suite for implementing high-performance gameplay and game logic [2]. Moreover, Unity supports 3D and 2D development with functionalities and features for the specific needs that are further focused on categories.

Similarly, [3] has quoted that the navigation system has also been included in Unity that

permits to create NPCs that logically move around the world of gaming. Navigation meshes have been used in the system that is automatically created from the scene geometry, or sometimes even from the dynamic obstacles, to change the character's navigation at runtime. Unity Prefabs that are known as the pre-configured objects of gaming provide the workflow flexibility and efficiency that permit in the confident working without nerve-wracking about the time-consuming errors [4]. According to [5] it has been found that Unity built-in UI system permits to create the UI intuitively and smartly with less consumption of time. Unity takes Box2D advantage the novel DOTS-based NVIDIA PhysX and Physics system that aids in the provision of high-performance and high realistic gameplay. Developers can extend the Editor of Unity with tools need to match the workflow of teams. It also supports the creation and customizing of the extensions that feature many possessions, extensions, and tools for the sake of speediness of the projects.

In addition, some of the key features of unity are related to the simple workflow that allows developers to quickly combine scenes in a workspace referred to as (intuitive editor). It also supports creating high quality games like AAA images, high definition sound, and action at full speed with no gaps on the screen [6]. Also, some extra features are [7]access the components, coroutine and return types, creating and destroying GameObjects, dealing with vector variables and timing variables, events for GameObject, and physics-oriented events. Primarily, Unity supports scripting in C, and there are two ways to design C scripts in Unity: object-oriented design, which is the more traditional and widely used approach, and data-oriented design, which is now possible in Unity [8].

Chapter 3

Methodology

Unity is a cross-platform game engine developed by Unity Technologies, first announced and released in June 2005 at Apple Inc.'s Worldwide Developers Conference as a Mac OS X exclusive game engine. The engine has since been gradually extended to support a variety of desktop, mobile, console and virtual reality platforms. It is particularly popular for iOS and Android mobile game development and used for games such as Pokemon Go, Monument Valley, Call of Duty: Mobile, Beat Saber and Cup-head. The engine can be used to create three-dimensional (3D) and two-dimensional (2D) games, as well as interactive simulations and other experiences.The engine has been adopted by industries outside video gaming, such as film, automotive, architecture, engineering, construction, and the United States Armed Forces.This project is entirely built on Unity engine.

3.1 Introduction

Captain-G is a 3D First-Person Shooter(FPS) game developed with Unity Engine. First-person shooter (FPS) is a sub-genre of shooter video games centered on guns and other weapon-based combat in a first-person perspective, with the player experiencing the action through the eyes of the protagonist and controlling the player character in a three-dimensional space. The genre shares common traits with other shooter games. Shooter video games or shooters are a subgenre of action video games where the focus is almost entirely on the defeat of the character's enemies using the weapons given to the player. First-person shooters relies on a first-person point of view with which the player experiences the action through the eyes of the character.

They differ from third-person shooters in that, in a third-person shooter, the player can see the character they are controlling (usually from behind, or above). The primary design focus is combat, mainly involving firearms or other types of long range weapons. Unity gives users the ability to create games and experiences in both 2D and 3D, and the engine offers a primary scripting API in C#, for both the Unity editor in the form of plugins, and games themselves, as well as drag and drop functionality. For 3D games, Unity allows specification of texture compression, mipmaps, and resolution settings for each platform that the game engine supports, and provides support for bump mapping, reflection mapping, parallax mapping, screen space ambient occlusion (SSAO), dynamic shadows using shadow maps, render-to-texture and full-screen post-processing effects. The level is designed by using ProGrids in unity. ProGrids is an essential tool that is used to help place objects with ease and precision. We can add game objectives in unity easily. The objective system works simply by adding GameObjects with an Objective component on them coupled with a specific component like ”ObjectiveKillEnemies” or ”ObjectiveReachPoint”.

3.2 First Person Shooter(FPS)

First-person shooter (FPS) is a video game genre centered around gun and other weapon-based combat in a first-person perspective i.e. the player experiences the action through the eyes of the protagonist. The genre shares common traits with other shooter games, which in turn makes it fall under the heading action game. Since the genre’s inception, advanced 3D and pseudo-3D graphics have challenged hardware development, and multiplayer gaming has been integral.

The first-person shooter genre has been traced as far back as Maze War, development of which began in 1973, and 1974’s Spasim. Later, and after more playful titles like MIDI Maze in 1987, the genre coalesced into a more wantonly violent form with 1992’s Wolfenstein 3D, which has been credited with creating the genre’s basic archetype, that subsequent titles were based upon. One such title, and the progenitor of the genre’s wider mainstream acceptance and popularity was Doom, perhaps one of the most influential games in this genre. For many years, the term Doom clone was used to designate this genre due to Doom’s influence. In 1998’s Half-Life—along with its 2004 sequel Half-Life 2—enhanced the narrative and puzzle

elements. In 1999, the Half-Life mod Counter-Strike was released and, together with Doom, is perhaps one of the most influential first-person shooters. GoldenEye 007, released in 1997, was a landmark first-person shooter for home consoles, while the Halo series heightened the console's commercial and critical appeal as a platform for first-person shooter titles. In the 21st century, the first-person shooter is the most commercially viable video game genre, and in 2016, shooters accounted for over 27% of all video game sales. Several first-person shooters have been popular games for sports and competitive gaming competitions as well.

3.3 Modules

3.3.1 Module - 1 : Prototype

A sample scene with templates describing the main theme or story of the game. The scene will contain:

- Main character(Hero) and weapons
- World or terrain
- Enemies
- Other game objects

3.3.2 Module - 2: Design

Game Design

Like most shooter games, first-person shooters involve an avatar, one or more ranged weapons, and a varying number of enemies. Because they take place in a 3D environment, these games tend to be somewhat more realistic than 2D shooter games, and have more accurate representations of gravity, lighting, sound and collisions. First-person shooters played on personal computers are most often controlled with a combination of a keyboard and mouse. This system has been claimed as superior to that found in console games, which frequently use two analog sticks: one used for running and sidestepping, the other for looking and aiming. It is common to display the character's hands and weaponry in the main view, with a head-up

display showing health, ammunition and location details. Often, it is possible to overlay a map of the surrounding area.

Combat and power-ups

First-person shooters often focus on action game play, with fast-paced and bloody firefights, though some place a greater emphasis on narrative, problem-solving and logic puzzles. In addition to shooting, melee combat may also be used extensively. In some games, melee weapons are especially powerful, a reward for the risk the player must take in maneuvering his character into close proximity to the enemy. In other situations, a melee weapon may be less effective, but necessary as a last resort. ”Tactical shooters” are more realistic, and require teamwork and strategy to succeed, the player often commands a squad of characters, which may be controlled by the game or by human teammates.

First-person shooters typically give players a choice of weapons, which have a large impact on how the player will approach the game. Some game designs have realistic models of actual existing or historical weapons, incorporating their rate of fire, magazine size, ammunition amount, recoil and accuracy. Other first-person shooter games may incorporate imaginative variations of weapons, including future prototypes, ”alien technology” scenario defined weaponry, and/or utilizing a wide array of projectiles, from industrial labor tools to laser, energy, plasma, rocket and grenade launchers or crossbows. These many variations may also be applied to the tossing animations of grenades, rocks, spears and the like. Also, more unconventional modes of destruction may be employed from the viewable user’s hands such as flames, electricity, telekinesis or other supernatural constructions. However, designers often allow characters to carry varying multiples of weapons with little to no reduction in speed or mobility, or perhaps more realistically, a pistol or smaller device and a long rifle or even limiting the player to only one weapon at a time. There are often options to trade up, upgrade or swap out in most games. Thus, the standards of realism vary between design elements. The protagonist can generally be healed and re-armed by means of items such as first aid kits, simply by walking over them. Some games allow players to accumulate experience points similar to those found in role-playing games, which can unlock new weapons and abilities.

Level Design

First-person shooters may be structurally composed of levels, or use the technique of a continuous narrative in which the game never leaves the first-person perspective. Others feature large sandbox environments, which are not divided into levels and can be explored freely. In first-person shooters, protagonists interact with the environment to varying degrees, from basics such as using doors, to problem solving puzzles based on a variety of interactive objects. In some games, the player can damage the environment, also to varying degrees: one common device is the use of barrels containing explosive material which the player can shoot, destroying them and harming nearby enemies. Other games feature environments which are extensively destructible, allowing for additional visual effects. The game world will often make use of science fiction, historic (particularly World War II) or modern military themes, with such antagonists as aliens, monsters, terrorists and soldiers of various types. Games feature multiple difficulty settings; in harder modes, enemies are tougher, more aggressive and do more damage, and power-ups are limited. In easier modes, the player can succeed through reaction times alone; on more difficult settings, it is often necessary to memorize the levels through trial and error.

Other design aspects:

- Terrain and world design
- Character design and customization
- Weapon design and modifications

3.3.3 Module - 3: Game Mechanics and Animation

Animation in game

Computer animation is the process used for generating animated images. The more general term computer-generated imagery (CGI) encompasses both static scenes and dynamic images, while computer animation only refers to the moving images. Modern computer animation usually uses 3D computer graphics, although 2D computer graphics are still used for stylistic, low bandwidth, and faster real-time renderings. Sometimes, the target of the animation is the computer itself, but sometimes film as well.

To create the illusion of movement, an image is displayed on the computer monitor and repeatedly replaced by a new image that is similar to it, but advanced slightly in time (usually at a rate of 24, 25 or 30 frames/second). This technique is identical to how the illusion of movement is achieved with television and motion pictures. For 3D animations, all frames must be rendered after the modeling is complete.

Other game mechanics:

- Player movement and actions
- Game physics
- Key bindings and game controls

3.3.4 Module - 4: Game AI

A.I. in Game

AI in gaming refers to responsive and adaptive video game experiences. These AI-powered interactive experiences are usually generated via non-player characters, or NPCs, that act intelligently or creatively, as if controlled by a human game-player. AI is the engine that determines an NPC's behavior in the game world.

While AI in some form has long appeared in video games, it is considered a booming new frontier in how games are both developed and played. AI games increasingly shift the control of the game experience toward the player, whose behavior helps produce the game experience.AI procedural generation, also known as procedural storytelling, in game design refers to game data being produced algorithmically rather than every element being built specifically by a developer. AI implementations in this project are the following:

- Enemy behavior
- Combat AI
- Player recognition
- Hunting

3.3.5 Module - 5: User interface

The game UI works on basic Canvas which was created for the intro scene (IntroMenu). This intro scene was further duplicated and modified to create other canvases, where the static elements are visible on the screen. During gameplay it floats on the screen.

3.3.6 Main Menu and Game Controls

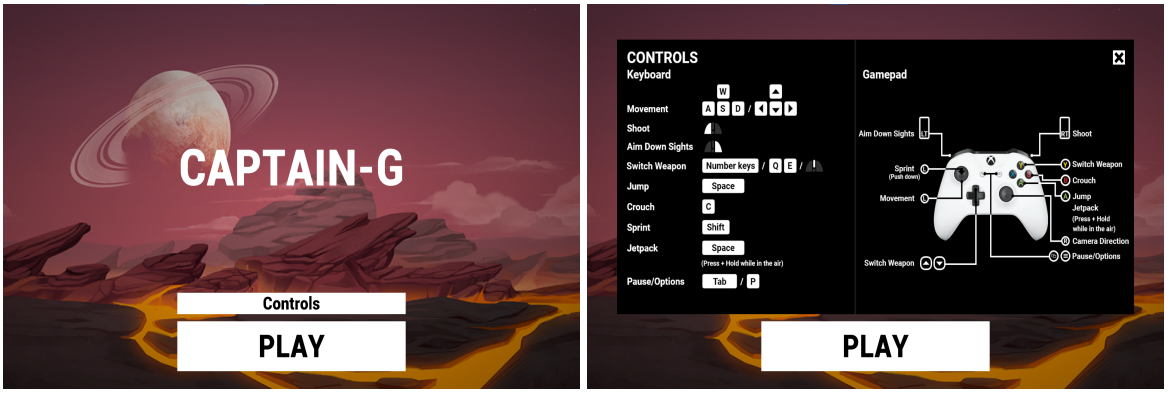

Figure 3.1: Main menu, Game controls

3.3.7 Gameplay Preview and Pause Menu

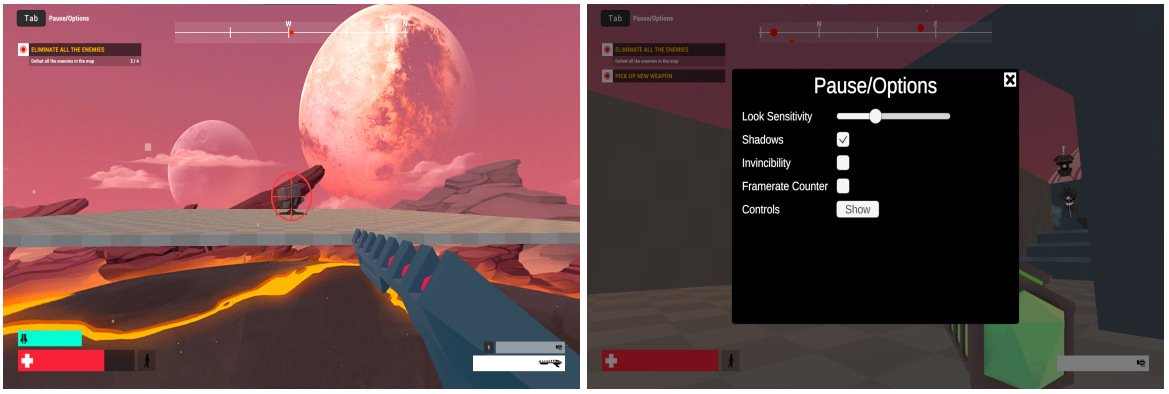

Figure 3.2: Gameplay Preview, Pause Menu

3.3.8 Win and Lose Scenes

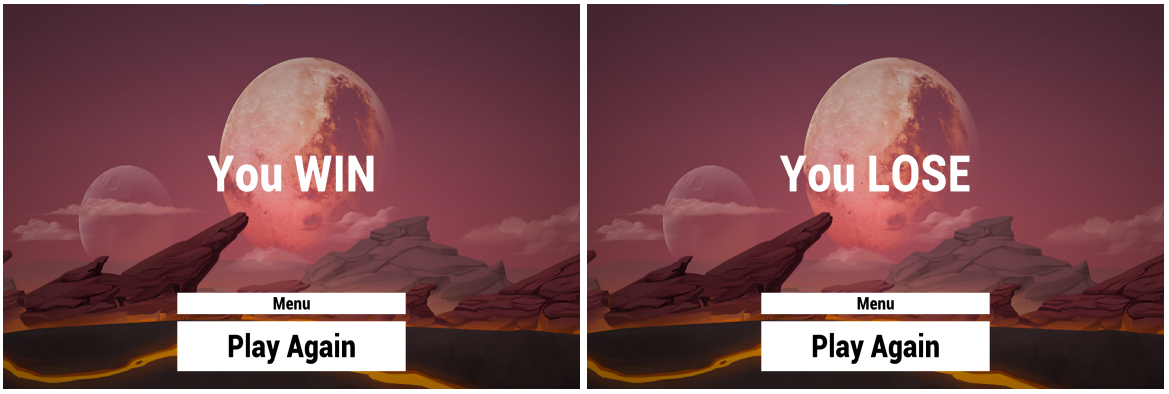

Figure 3.3: win scene, lose scene

3.4 Developing Environment

3.4.1 Hardware Requirements

- Processor - Intel Core i5 (min)
- Clock Speed - 1.5 GHz (min)
- RAM - 4 GB (min)
- Hard Disk - 50 GB or (min)
- GPU - 1 GB (min)

3.4.2 Software Requirements

- Operating System - Windows 7 or above
- Game Engine - Unity
- Programming Language - C
- IDE - Visual Studio 2019
- SFX - Audacity

- Modeling - Blender (if necessary
- Texturing - Photoshop (if necessary)

3.5 Introduction to the Platform

Unity is an all-purpose game engine that supports 2D and 3D graphics, drag and drop functionality and scripting through C#.

Two other programming languages were supported: Boo, which was deprecated with the release of Unity 5 and UnityScript(JavaScript) which was deprecated in August 2017 after the release of Unity 2017.1. The engine targets the following graphics APIs: Direct3D on Windows and Xbox One; OpenGL on Linux, macOS, and Windows; OpenGL ES on Android and iOS; WebGL on the web; and proprietary APIs on the video game consoles.

Additionally, Unity supports the low-level APIs Metal on iOS and macOS and Vulkan on Android, Linux, and Windows, as well as Direct3D 12 on Windows and Xbox One. Within 2D games, Unity allows importation of sprites and an advanced 2D world renderer.

For 3D games, Unity allows specification of texture compression and resolution settings for each platform that the game engine supports, and provides support for bump mapping, reflection mapping, parallax mapping, screen space ambient occlusion (SSAO), dynamic shadows using shadow maps, render-to-texture and full-screen post-processing effects. Unity also offers services to developers, these are: Unity Ads, Unity Analytics, Unity Certification, Unity Cloud Build, Unity Every play, Unity IAP, Unity Multiplayer, Unity Performance Reporting and Unity Collaborate.

3.5.1 Game Engine

A game engine is a software development environment designed for people to build video games. Developers use them to create games for consoles, mobile devices, and personal computers. The core functionality typically provided by a game engine includes a rendering engine (”renderer”) for 2D or 3D graphics, a physics engine or collision detection (and collision response), sound, scripting, animation, artificial intelligence, networking, streaming, memory management, threading, localization support, scene graph, and may include video

support for cinematics. The process of game development is often economized, in large part, by reusing/adapting the same game engine to create different games or to make it easier to port games to multiple platforms.

3.5.2 Types of Game Engine

XNA Game Studio (Microsoft)

Microsoft XNA is one of the best opportunities in the game industry right now. It's a .NET programming package that lets you make any small to medium sized game you're capable of coding, and then it allows you to share your creation with the world on Xbox Live. How cool is that? It's like a cheat code or a cool friend who lets you bypass the long line at a club; the exposure of Xbox Live isn't a free perk but it's something that's hard for the others on this list to match.

Unreal Engine (Epic Games)

This is pretty much the undisputed best game engine in the current era of the game industry. With literally hundreds of games – even MMO games – that have used it in the past, the Unreal Engine is capable of creating any game for any genre with any budget. While it's by no means something an amateur should tackle solo to make the game of their dreams, it is however a great tool for creating the 3D level of your dreams.

Unreal Tournament 2004 comes with UnrealEd – the Unreal Engine's level editor – bundled in for free, and after one or two nights of reading and watching tutorials online you'll be able to create your own 3D level that you and your friends can play online. It lets you use hundreds of objects and textures from the game to create your own levels, but importing your own data is also an option. Those interested in just the FPS angle should also check out Valve's Hammer Editor, which is another valuable game bundle worth having and knowing.

Unity (Unity Technology)

With the humongous rise of handheld electronics in recent years, the usage of Unity has soared and has even stolen a bit of thunder from the rest on this list. It's similar to the Torque and Blender game engines – which are also really worth checking out – and it's capable of creating

games for PC, Mac, Wii, Xbox 360, and PS3, but its popularity in the past year has been largely due to its use in developing iPad and iPhone games. It's so popular right now that I actually plan to take my own advice and learn Unity in the coming months to stay up to date with the game industry!

3.6 Unity 3D Game Engine

Unity is a cross-platform game engine developed by Unity Technologies. which is primarily used to develop video games and simulations for computers, consoles and mobile devices. First announced only for OS X, at Apple's Worldwide Developers Conference in 2005, it has since been extended to target 27 platforms.

Six major versions of Unity have been released. At the 2006 WWDC show, Apple named Unity as the runner up for its Best Use of Mac OS X Graphics category.

Features of Unity:-

- 2D
- 3D
- Graphics
- Physics
- Scripting
- Multiplayer and Networking
- Audio
- Animation
- Timeline
- UI
- Navigation and Path finding
- Virtual Reality
- Open-source repositories
- Asset Store Publishing
- Platform-specific
- Experimental

3.7 C# or C Sharp

C# is a modern, general-purpose, object-oriented programming language developed by Microsoft and approved by European Computer Manufacturers Association (ECMA) and International Standards Organization (ISO). C# was developed by Anders Hejlsberg and his team

during the development of .Net Framework. C# is designed for Common Language Infrastructure (CLI), which consists of the executable code and runtime environment that allows use of various high-level languages on different computer platforms and architectures.

The following reasons make C# a widely used professional language :

- It is a modern, general-purpose programming language
- It is object oriented.
- It is component oriented.
- It is easy to learn.
- It is a structured language.
- It produces efficient programs.
- It can be compiled on a variety of computer platforms.

3.8 VISUAL STUDIO 2019

The Visual Studio interactive development environment (IDE) is a creative launching pad that you can use to view and edit nearly any kind of code, and then debug, build, and publish apps for Android, iOS, Windows, the web, and the cloud. There are versions available for Mac and Windows. This topic introduces you to the features of the Visual Studio IDE. We'll walk through some things you can do with Visual Studio and how to install and use it, create a simple project, get pointers on debugging and deploying code, and take a tour of the various tool windows.

With Visual Studio Tools for Unity (VSTU), you can use Visual Studio to write game and editor scripts in C and then use its powerful debugger to find and fix errors. The latest release of VSTU includes syntax coloring for Unity's ShaderLab shader language, better debugger visualizations, and improved code generation for the MonoBehavior wizard. VSTU also brings your Unity project files, console messages, and the ability to start your game into Visual Studio so you can spend less time switching to and from the Unity Editor while writing code.Visual Studio Tools for Unity registers a log callback so it can stream the Unity console

to Visual Studio. If you have editor scripts that log information, you can plug them into the same callback to send your messages to Visual Studio.

3.9 Agile Methodology

The Agile methodology is a way to manage a project by breaking it up into several phases. It involves constant collaboration with stakeholders and continuous improvement at every stage. Once the work begins, teams cycle through a process of planning, executing, and evaluating. Continuous collaboration is vital, both with team members and project stakeholders.This project was completed in 2 sprints with 5 modules.

3.9.1 Sprint 1

- Implementing Player Character
- Creation of enemy characters
- World and terrain design
- Scripting
- Basic UI

3.9.2 Sprint 2

- Key bindings
- AI for enemy Characters
- Final UI

3.9.3 User Story

A key component of agile software development is putting people first, and user-stories put actual end users at the center of the conversation. Stories use non-technical language to provide context for the development team and their efforts. After reading a user story, the team knows why they are building what they're building and what value it creates. A user story is a tool used in agile software development to capture a description of a software feature from an end user perspective. The user story describes the type of user, what they want and why. A user story helps to create a simplified description of a requirement. User stories are one of the core components of an agile program. They help provide a user-focused framework for daily work which drives collaboration, creativity, and a better product overall. The user story of system is given in the below figure.

User Story ID	As a type of User	I want to <perform some task>	So that I can < Achieve Some Goal>
1	Player	Control the main character	Move and interact in the game world
2		See the enemies	Destroy the enemies
3		See the world and terrain	Move around the world
4		Navigation	Know where to go next
5		See the UI for main mode	Play the main mode
6		See type of enemies and see different weapons	Use different weapons on different enemies
7		View key bindings	Know the gaming controls
8		See final UI	

Figure 3.4: User Story

3.9.4 Product Backlog

A product backlog is a list of the new features, changes to existing features, bug fixes, infrastructure changes or other activities that a team may deliver in order to achieve a specific outcome.The product backlog is the single authoritative source for things that a team works on. That means that nothing gets done that isn't on the product backlog. Conversely, the presence of a product backlog item on a product backlog does not guarantee that it will be

delivered. It represents an option the team has for delivering a specific outcome rather than a commitment.It should be cheap and fast to add a product backlog item to the product backlog, and it should be equally as easy to remove a product backlog item that does not result in direct progress to achieving the desired outcome or enable progress toward the outcome. The Scrum Product Backlog is simply a list of all things that needs to be done within the project. It replaces the traditional requirements specification artifacts. These items can have a technical nature or can be user-centric e.g. in the form of user stories.The product backlog of the system is given below figure.

Sl.No	Priority <High / Medium / Low>	Size (Hours)	Sprint <#>	Status <Planned / In progress / Completed>	Release Date	Release Goal
1	High	6	1	Completed		Creation of player in first person perspective
2	High	6		Completed		Creation of enemy characters
3	High	4		Completed		Maps(Terrain, World)
4	High	6	2	Completed		World / Terrain design
5	High	2		Completed		Main mode (Scripting, UI)
6	Low	2		Incomplete		Time attack (Scripting, UI)
7	High	10	3	Completed		AI and Selectable weapons (Coding)
8	High	2	4	Completed		Difficulty levels
9	Low	2		Completed		Key Binding
10	High	10		Completed		Final UI

Figure 3.5: Product Backlog

3.9.5 Project Plan

A project plan that has a series of tasks laid out for the entire project, listing task duration, responsibility assignments, and dependencies. Plans are developed in this manner based on the assumption that the Project Manager, hopefully along with the team, can predict up front everything that will need to happen in the project, how long it will take, and who will be able to do it. Project plan is given below figure. The project has two sprints.

User Story ID	Task Name	Start Date	End Date	Days	Status
SPRINT 1					
1	Player Character	30/11/2021	03/12/2021	4	Completed
2	Creation of enemy characters	04/12/2022	11/12/2022	8	Completed
3	World / Terrain design	12/12/2022	27/12/2022	6	Completed
4	Navigation pane	28/12/2022	29/01/2022	2	Completed

Figure 3.6: Project Plan 1.1

User Story ID	Task Name	Start Date	End Date	Days	Status
SPRINT 2					
5	Main mode (Scripting, UI)	31/01/2022	03/02/2022	4	Completed
6	AI and Selectable weapons (Coding)	04/02/2022	12/02/2022	8	Completed
7	Key Binding	13/02/2022	15/02/2022	3	Completed
8	Final UI	15/02/2022	16/02/2022	1	Completed

Figure 3.7: Project Plan 1.2

3.9.6 Sprint backlog

The sprint backlog is a list of tasks identified by the Scrum team to be completed during the Scrum sprint. During the sprint planning meeting, the team selects some number of product backlog items, usually in the form of user stories, and identifies the tasks necessary to complete each user story. Most teams also estimate how many hours each task will take someone on the team to complete.

Backlog Item	Status & completion date	Original estimate in hours	Day 1	Day2	Day3	Day4	Day5	Day6	Day7	Day8	Day9	Day10	Day11	Day12	Day13	Day14
User story #1		hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs
Modelling (Player)	Completed	4	1	1	0	1	1	0	0	0	0	0	0	0	0	0
Script	Completed	2	0	0	0	0	0	1	0	1			0	0	0	0
Testing		Continuous	Yes													
User story #2	Completed															
Modelling (Enemies)	Completed	4	0		0	0	2	0	0	0	0	0	1	1	0	0
Script	Completed	2	0	0	0	0	0	1	0	0	0	0	1	0	0	0

Figure 3.8: Sprint backlog 1.1

Backlog Item	Status & completion date	Original estimate in hours	Day1	Day2	Day3	Day4	Day5	Day6	Day7	Day8	Day9	Day10	Day11	Day12	Day13	Day14
User story #3 Design (navigation)	Completed	4	1	1	1	1		0	0	0	0	0	0	0	0	0
User story #4 Design (World, terrain)	Completed	6	1	1	1	0	0	0	1	1	1	0	0	0	0	0
User story #5																
(UI, Scripting)	Completed	2	0	0	0	0	0	0	0	0	1	1	0	0	0	0
Testing	Completed	Continuous														

Figure 3.9: Sprint backlog 1.2

Backlog Item	Status & completion date	Original estimate in hours	Day 1	Day2	Day3	Day4	Day5	Day6	Day7	Day8	Day9	Day10	Day11	Day12	Day13	Day14
User story #6		hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrsE
Enemy (Design &Script-AI)	Completed	2	0	0	0	0	0	1	0	1			0	0	0	0
Testing	Completed	Continuous	Yes													
User story #7																
Scripting	Completed	10	2	0	2	0	1	1	1	0	2	0	0	0	1	

Figure 3.10: Sprint backlog 1.3

Backlog Item	Status & completion date	Original estimate in hours	Day1	Day2	Day3	Day4	Day5	Day6	Day7	Day8	Day9	Day10	Day11	Day12	Day13	Day14
User story #8		hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs
Script(final UI)	Completed	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0
Testing	Completed	Continuous	Yes													
Final Test	Completed	1	0	0	0	0	0	0	0	0	0	0	0	1	1	

Figure 3.11: Sprint backlog 1.4

3.9.7 Sprint Actual

Actual sprint backlog is what adequate sprint planning is actually done by project team there may or may not be difference in planned sprint backlog. The detailed sprint backlog (Actual) is given below.

Backlog Item	Status & completion date	Original estimate in hours	Day1	Day2	Day3	Day4	Day5	Day6	Day7	Day8	Day9	Day10	Day11	Day12	Day13	Day14
User story #1		hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs
Modelling (player)	Completed	1	1	1	1	2	0									
Script	Completed	2						1	0	1	1	2	0	0	0	0
Testing	Completed	Continuous	Yes													
User story #2																
Modelling (Enemies)	Completed	6											1	1		
Scripting (enemies)	Completed	4													0	2

Figure 3.12: Sprint 1 Actual 1.1

Backlog Item	Status & completion date	Original estimate in hours	Day1	Day2	Day3	Day4	Day5	Day6	Day7	Day8	Day9	Day10	Day11	Day12	Day13	Day14
User story #3		hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs
Modelling (world, terrain, navigation)	Completed	7	1	1	1	2	0									
Script	Completed	3						1	0	1	1	2	0	0	0	0
Testing	Completed	Continuous	Yes													
User story #4																
Modelling (level Design)	Completed	3											1	1		
Scripting(pickups and rewards)	Completed	2													0	2

Figure 3.13: Sprint 1 Actual 1.2

Backlog Item	Status & completion date	Original estimate in hours	Day1	Day2	Day3	Day4	Day5	Day6	Day7	Day8	Day9	Day10	Day11	Day12	Day13	Day14
User story #5		hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs
Key binding (player)			1	1	1	2	0									
Script	Completed	3						1	0	1	1	2	0	0	0	0
Testing	Completed	Continuous	Yes													
User story #6																
AI (Enemies)	Completed	6											1	1		
Weapons (Scripting)	Completed	6													0	2

Figure 3.14: Sprint 2 Actual 1.1

Backlog Item	Status & completion date	Original estimate in hours	Day1	Day2	Day3	Day4	Day5	Day6	Day7	Day8	Day9	Day10	Day11	Day12	Day13	Day14
User story #7		hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs	hrs
Final UI	2		1	1	1	2	0									
Final Testing	Completed	Continuous	Yes													

Figure 3.15: Sprint 2 Actual 1.2

Chapter 4

Results and Discussions

4.1 Results

When we open the executable file, the game loads up and displays the main menu which shows the title of the game and has the level environment as the background. There is two buttons on the Main Menu, Controls and PLAY(Figure 4.1: Main menu).

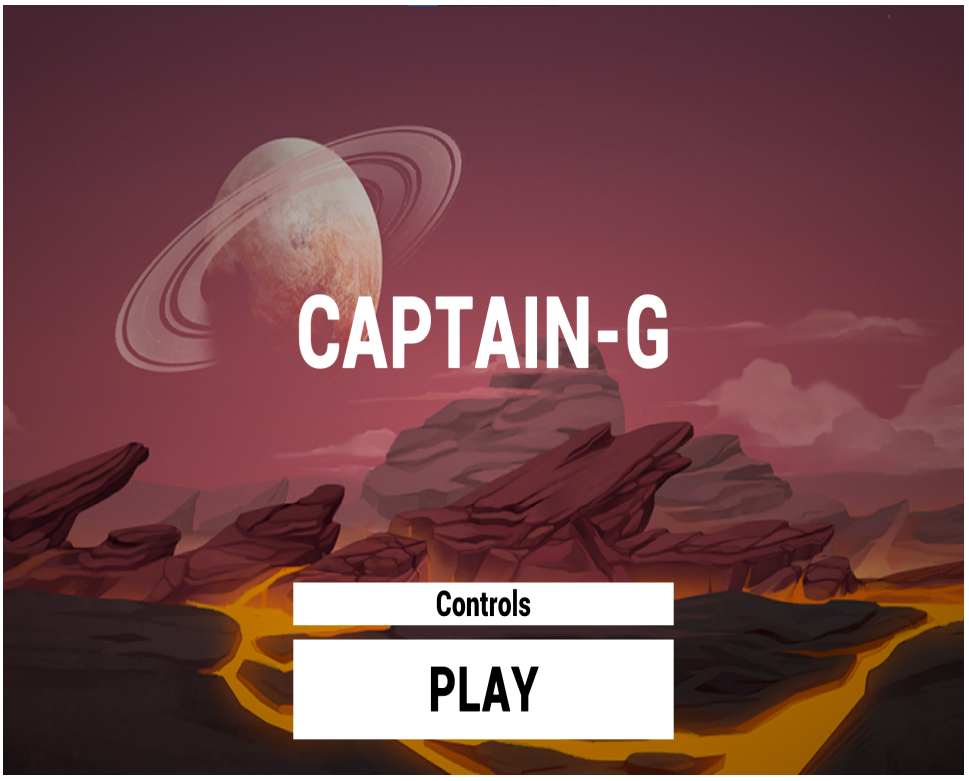

Figure 4.1: Main menu

When the Controls button is clicked, the player is redirected to the Controls Scene(Figure 4.2: Game controls) which displays the keybindings that the player must follow to play the game. There is controls for both keyboard + mouse combination and for a gamepad.

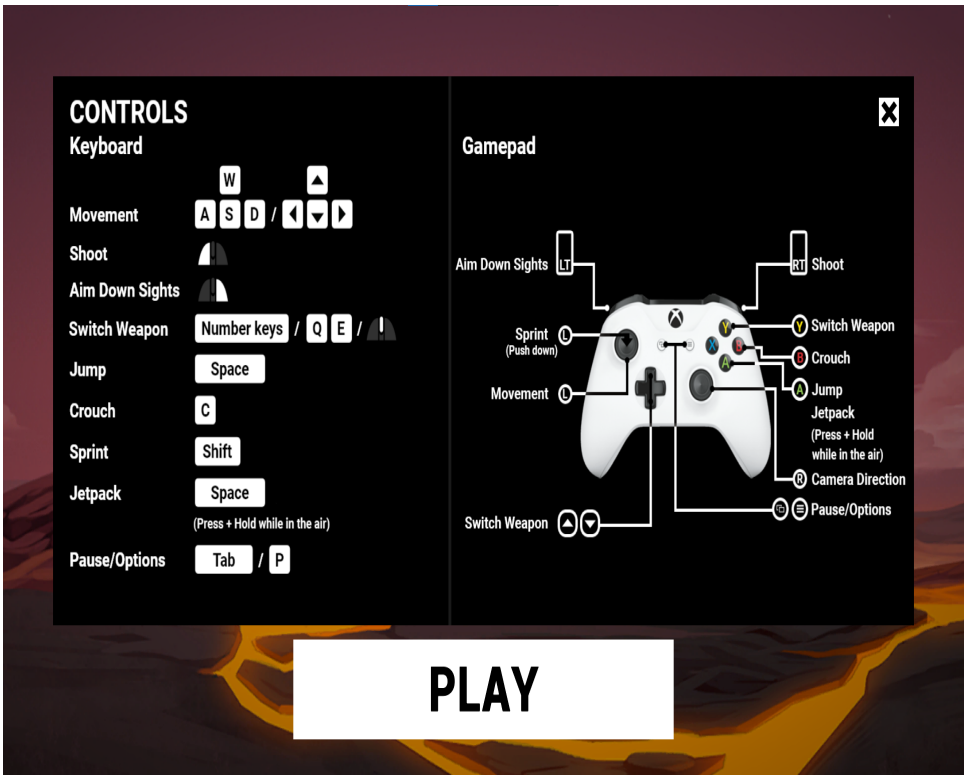

Figure 4.2: Game controls

Upon clicking the PLAY button the game level will Load and the player will be spawned in to the game world. The scene will contain all basic game UI components like player health meter, weapon overheat meter, navigation pane, pause/option and the mission objectives will be displayed on the left corner of the screen where the main objective is on the top and optional objectives below it(Figure 4.3: Game play beginning). The player can control the player using the assigned control buttons and move around the world and use the mouse or joystick buttons to shoot the enemies. The red dots on the navigation pane indicates the position of the enemies in the game world, it works as a digital compass with four directions and a middle line indicating the altitude of the game world.

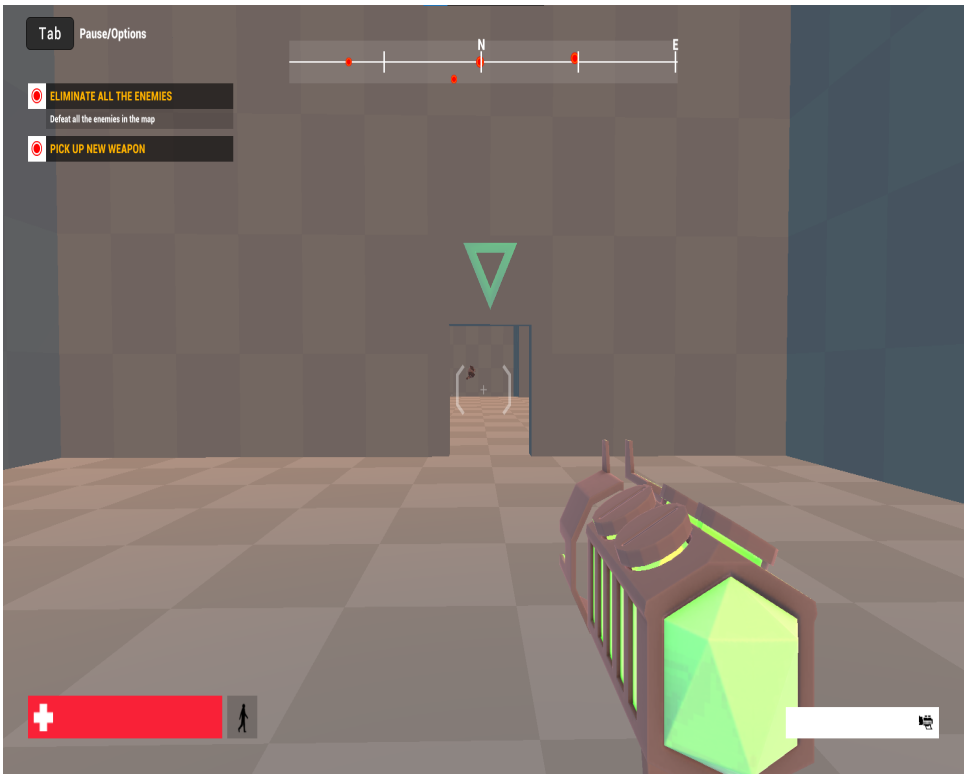

Figure 4.3: Game play beginning

When we try to approach an enemy, the enemy will detect our presence and will start shooting at us. An exclamation mark will be displayed on their head so that the player can know that they have detected us. This is done using the AI implementation on the enemy characters. There are two types of enemy characters in this level, Hover bots and Turrets. In the case of hover bots once you are detected then they will start to follow you anywhere you go until u destroy them(Figure 4.4: Hover Bots). This enemy AI is called hunting and is a famous implementation of AI in gaming. Turrets on the other hand are static bots which can detect our presence and shoot at us if we go near them but they can only rotate towards us as they are fixed on a stand(Figure 4.5: Turret). Both enemies have their own strengths and weaknesses. Hover bots can follow you and can move fast which makes them hard to shoot at, but they have less firing power and only deals less damage to us. Turrets are static but they have more rate of fire and can deal more damage to us in seconds.

This game has two weapons on the current level, first one is the default weapon where the player spawned with. This gun works like a machine gun and have a high rate of fire with less damage dealing. The gun will over heat on continuous use and will change color from green

Figure 4.4: Hover Bots

to yellow and stop working unless proper cooling time is given(Figure 4.6: Machine Gun).By clicking right mouse button you can aim down sight and will give you a zoomed view which will help you to shoot more accurately(Figure 4.6: (Figure 4.6: Aim down sight).

The second weapon is picked up from the game world as a side mission by roaming around and works like a sniper life. It has more firing power and high zooming level while aiming down sight. But only one shot can be take once ans the gun will take while to cool(Figure 4.7: Sniper rifle, Aim Down Sight). So both guns have their own advantages and disadvantages. This will make the gameplay more fun and gives a balanced challenge and reward system to the game. You can use the machine guns on the flying hover bots and use the sniper rifles on the turrets. We can switch between the two weapons using the mouse wheel or using the the number keys 1 and 2.

There is a jetpack in the game which u will get as a loot item after you destroy a turret. You can use this jetpack to reach places where u cannot reach normally. The jetpack has only limited amount of fuel where u can only use for some seconds and it will regenerate soon after some time. There will be a fuel indicator on the left bottom side of the screen indicating

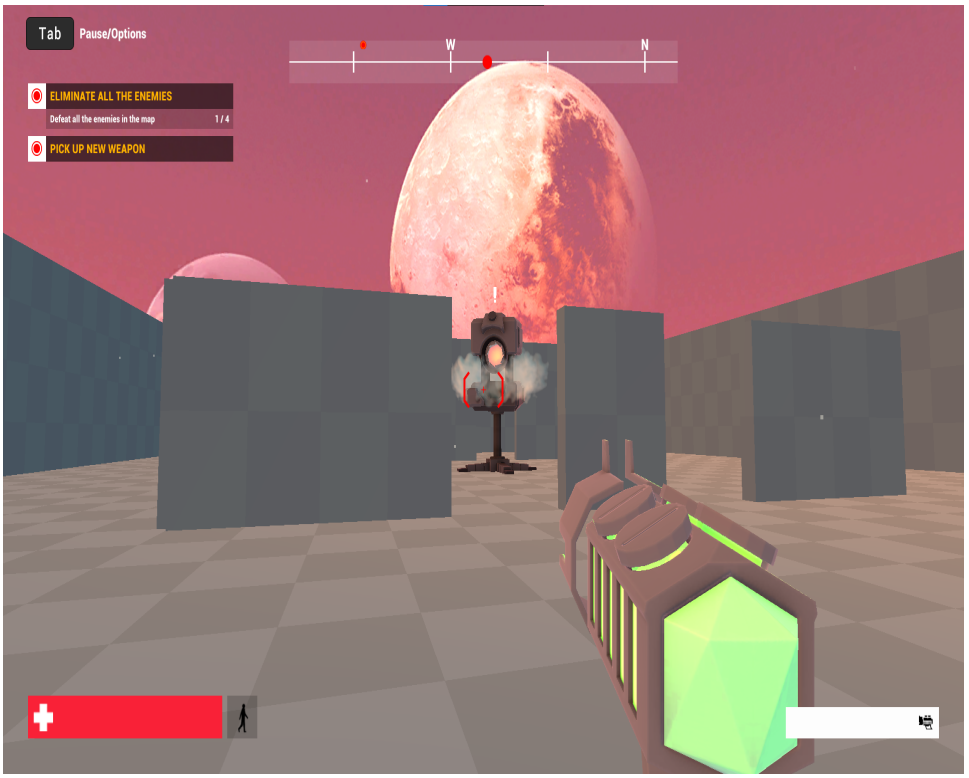

Figure 4.5: Turret

this. It has also a beautiful blue trail where the player can see when the looking down(Figure 4.8: Jetpack). Finally, after eliminating all the enemies and completing all required sub tasks the level will complete and the win scene will display, from where you can replay the game one more time using the replay button or you can go to the main menu by clicking the Menu button. But if you get killed by any one of the enemies or fall down from the walkable mesh, you will die and the lose scene will display, from where u can try again(Figure 4.9: win scene, lose scene).

Milestones completed:

- Prototype build was completed successfully
- A playable level was completed successfully
- UI was designed completely
- AI implementations was incorporated to the enemies successfully
- loot items was added successfully

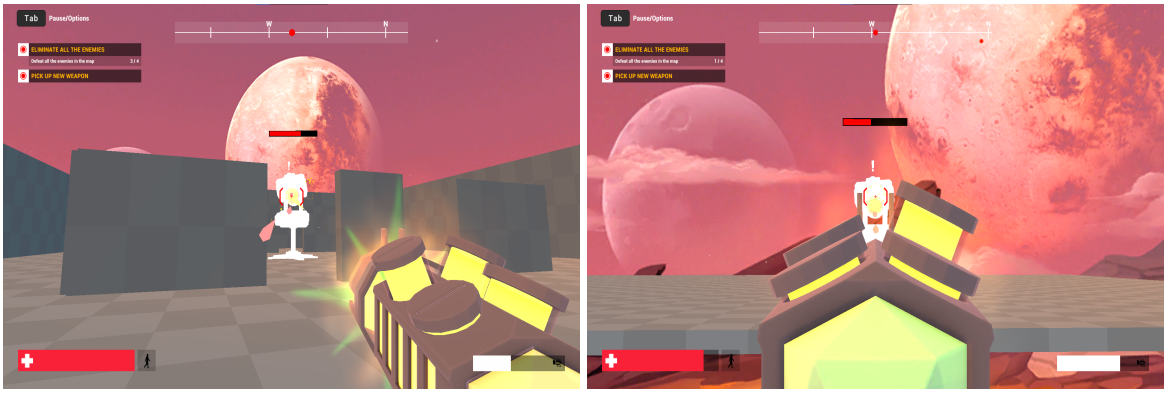

Figure 4.6: Machine Gun, Aim Down Sight

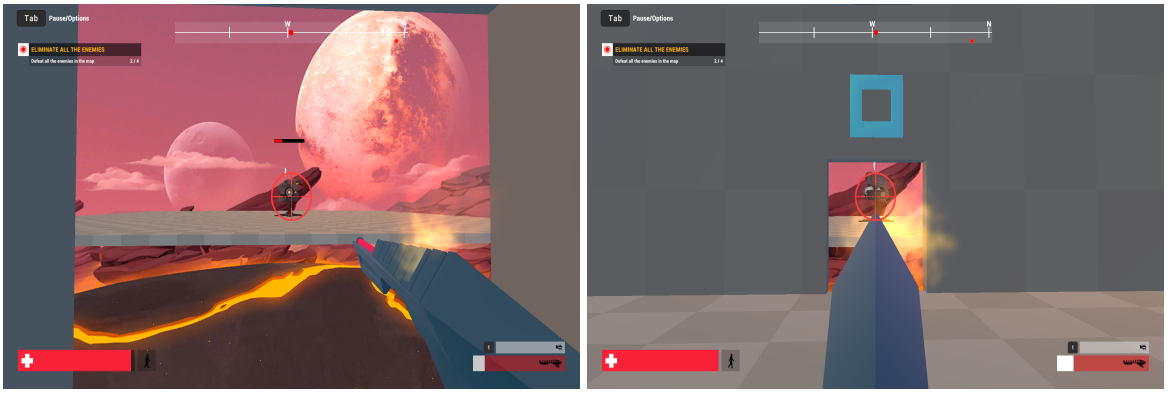

Figure 4.7: Sniper rifle, Aim Down Sight

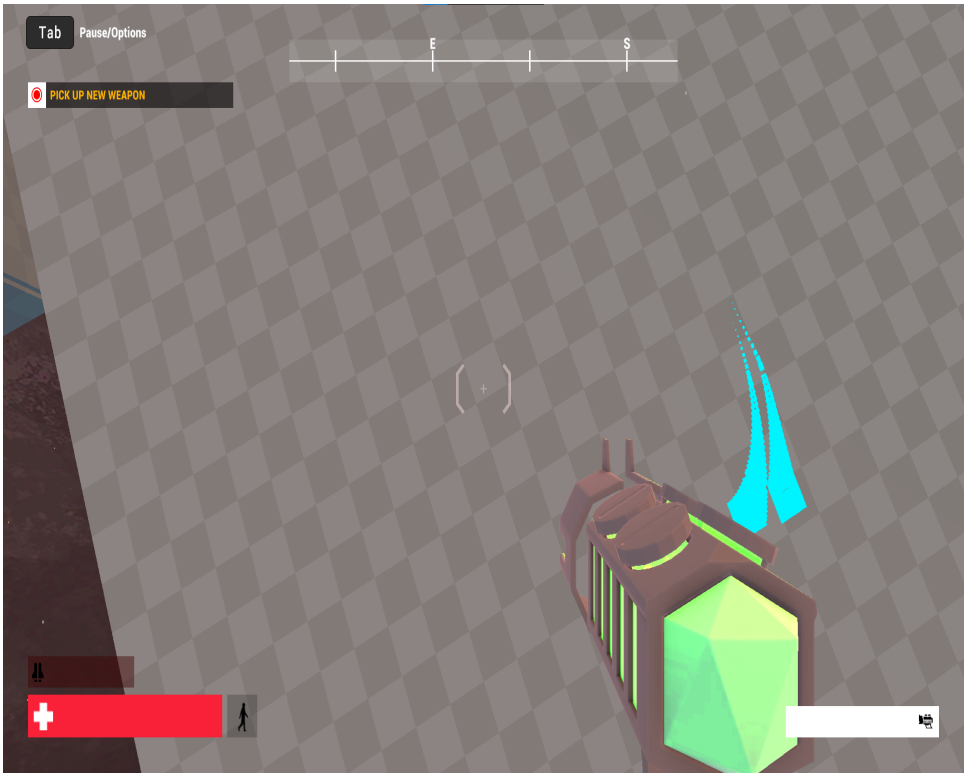

Figure 4.8: Jetpack

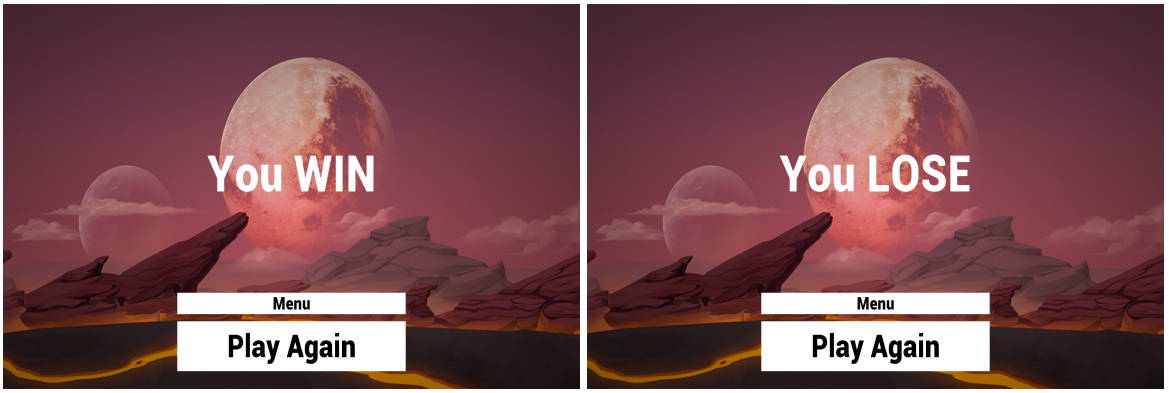

Figure 4.9: win scene, lose scene

Chapter 5

Conclusions

The Game was tested by over twenty users, who all reported that they thoroughly enjoyed the game. It was observed that there was approximately equal number of players who were able to kill the enemy. This suggests that not only did the game provide an entertaining gaming experience, it also provided a reasonably engaging and challenging gameplay.

In general, it can be concluded that the Unity platform supported efficient development of the First person Shooting Game. The Unity platform supports implementing the player's movement on the terrain with the keyboard and mouse , the physics engine, and vector calculation functions, all of which are not available if the implementation was done using traditional AI search techniques.

With these Unity components, We are able to implement the enemy's search for the player with less effort and more efficiently, and the prefab robotic characters can successfully imitate robotic shooting behaviours.

5.1 Future Enhancements

- Improved graphics
- Smoother movement and more responsive game controls
- Improved frame per second
- An entertaining story mode

- Improved and solid level design and world design
- More balanced challenge and reward
- More levels.
- New Weapons.
- New enemies
- Multiplayer facility for both online and offline gaming.
- Interactive objects and sub missions
- Cross Platform multiplayer
- Ally robots

References

[1] **Haas, John K**. "A history of the unity game engine." (2014) , [Online]. Available: https://digital.wpi.edu/downloads/2f75r821k?locale=en.

[2] **Canossa, Alessandro.**"Interview with Nicholas Francis and Thomas Hagen from Unity Technologies." In-Game Analytics, pp. 137-142. Springer, London, 2013.

[3] **Becker-Asano, Christian, Felix Ruzzoli, Christoph Hölscher, and Bernhard Nebel.** "A multi-agent system based on unity 4 for virtual perception and wayfinding." Transportation Research Procedia 2: 452-455.

[4] **Goldstone, Will.**. Unity 3. x game development essentials. Packt Publishing Ltd, 2011.

[5] **Patil, Pratik P., and Ronald Alvares.**."Cross-platform Application Development using Unity Game Engine." Int. J 3, no. 4

[6] **Andrade, António.**""Game engines: a survey." EAI Endorsed Transactions on Serious Games 2, no. 6

[7] **Borg, Markus, Vahid Garousi, Anas Mahmoud, Thomas Olsson, and Oskar Stalberg.**"Video Game Development in a Rush: A Survey of the Global Game Jam Participants." IEEE Transactions on Games

[8] **Malete, Thabo N., Kabo Moruti, Tsaone Swaabow Thapelo, and Rodrigo S. Jamisola** a. "EEG-based Control of a 3D Game Using 14-channel Emotiv Epoc+." In 2019 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM), pp.463-468. IEEE,2019.

[9] Unity Real-Time Development Platform, Available: https://unity.com/.

[10] Unity Learn, Available: https://learn.unity.com/.

[11] Unity Developer tools and Resouces, Available: https://unity.com/developer-tools.

[12] Unity 2D 3D Game Creation Solutions Services, Available: https://unity.com/solutions.

[13] Unity Asset Store, Available: https://assetstore.unity.com/.

[14] C documentation, Available: https://docs.microsoft.com/en-us/dotnet/csharp/.

[15] Learn to code in Visual Studio, Available: https://visualstudio.microsoft.com/vs/getting-started/.

[16] Visual Studio documentation, Available: https://docs.microsoft.com/en-us/visualstudio/windows/?view=vs-2022.

Appendix

Source Code

EnemyMobile.cs

```
using Unity.FPS.Game;
using UnityEngine;

namespace Unity.FPS.AI
{
   [RequireComponent(typeof(EnemyController))]
   public class EnemyMobile : MonoBehaviour
   {
      public enum AIState
      {
         Patrol,
         Follow,
         Attack,
      }

      public Animator Animator;

      [Tooltip("Fraction of the enemy's attack range at which it will stop moving towards target while attacking")]
      [Range(0f, 1f)]
      public float AttackStopDistanceRatio = 0.5f;

      [Tooltip("The random hit damage effects")]
      public ParticleSystem[] RandomHitSparks;

      public ParticleSystem[] OnDetectVfx;
      public AudioClip OnDetectSfx;

      [Header("Sound")] public AudioClip MovementSound;
      public MinMaxFloat PitchDistortionMovementSpeed;

      public AIState AiState { get; private set; }
      EnemyController m_EnemyController;
      AudioSource m_AudioSource;

      const string k_AnimMoveSpeedParameter = "MoveSpeed";
      const string k_AnimAttackParameter = "Attack";
      const string k_AnimAlertedParameter = "Alerted";
      const string k_AnimOnDamagedParameter = "OnDamaged";

      void Start()
      {
```

```
        m_EnemyController = GetComponent<EnemyController>();
        DebugUtility.HandleErrorIfNullGetComponent<EnemyController, EnemyMobile>(m_EnemyController, this,
            gameObject);

        m_EnemyController.onAttack += OnAttack;
        m_EnemyController.onDetectedTarget += OnDetectedTarget;
        m_EnemyController.onLostTarget += OnLostTarget;
        m_EnemyController.SetPathDestinationToClosestNode();
        m_EnemyController.onDamaged += OnDamaged;

        // Start patrolling
        AiState = AIState.Patrol;

        // adding a audio source to play the movement sound on it
        m_AudioSource = GetComponent<AudioSource>();
        DebugUtility.HandleErrorIfNullGetComponent<AudioSource, EnemyMobile>(m_AudioSource, this, gameObject);
        m_AudioSource.clip = MovementSound;
        m_AudioSource.Play();
    }

    void Update()
    {
        UpdateAiStateTransitions();
        UpdateCurrentAiState();

        float moveSpeed = m_EnemyController.NavMeshAgent.velocity.magnitude;

        // Update animator speed parameter
        Animator.SetFloat(k_AnimMoveSpeedParameter, moveSpeed);

        // changing the pitch of the movement sound depending on the movement speed
        m_AudioSource.pitch = Mathf.Lerp(PitchDistortionMovementSpeed.Min, PitchDistortionMovementSpeed.Max,
            moveSpeed / m_EnemyController.NavMeshAgent.speed);
    }

    void UpdateAiStateTransitions()
    {
        // Handle transitions
        switch (AiState)
        {
            case AIState.Follow:
                // Transition to attack when there is a line of sight to the target
                if (m_EnemyController.IsSeeingTarget && m_EnemyController.IsTargetInAttackRange)
                {
                    AiState = AIState.Attack;
                    m_EnemyController.SetNavDestination(transform.position);
                }

                break;
            case AIState.Attack:
                // Transition to follow when no longer a target in attack range
                if (!m_EnemyController.IsTargetInAttackRange)
                {
                    AiState = AIState.Follow;
                }

                break;
        }
    }

    void UpdateCurrentAiState()
    {
        // Handle logic
```

```
        switch (AiState)
        {
            case AIState.Patrol:
                m_EnemyController.UpdatePathDestination();
                m_EnemyController.SetNavDestination(m_EnemyController.GetDestinationOnPath());
                break;
            case AIState.Follow:
                m_EnemyController.SetNavDestination(m_EnemyController.KnownDetectedTarget.transform.position);
                m_EnemyController.OrientTowards(m_EnemyController.KnownDetectedTarget.transform.position);
                m_EnemyController.OrientWeaponsTowards(m_EnemyController.KnownDetectedTarget.transform.position);
                break;
            case AIState.Attack:
                if (Vector3.Distance(m_EnemyController.KnownDetectedTarget.transform.position,
                        m_EnemyController.DetectionModule.DetectionSourcePoint.position)
                    >= (AttackStopDistanceRatio * m_EnemyController.DetectionModule.AttackRange))
                {
                    m_EnemyController.SetNavDestination(m_EnemyController.KnownDetectedTarget.transform.position);
                }
                else
                {
                    m_EnemyController.SetNavDestination(transform.position);
                }

                m_EnemyController.OrientTowards(m_EnemyController.KnownDetectedTarget.transform.position);
                m_EnemyController.TryAtack(m_EnemyController.KnownDetectedTarget.transform.position);
                break;
        }
    }

    void OnAttack()
    {
        Animator.SetTrigger(k_AnimAttackParameter);
    }

    void OnDetectedTarget()
    {
        if (AiState == AIState.Patrol)
        {
            AiState = AIState.Follow;
        }

        for (int i = 0; i < OnDetectVfx.Length; i++)
        {
            OnDetectVfx[i].Play();
        }

        if (OnDetectSfx)
        {
            AudioUtility.CreateSFX(OnDetectSfx, transform.position, AudioUtility.AudioGroups.EnemyDetection, 1f);
        }

        Animator.SetBool(k_AnimAlertedParameter, true);
    }

    void OnLostTarget()
    {
        if (AiState == AIState.Follow || AiState == AIState.Attack)
        {
            AiState = AIState.Patrol;
        }

        for (int i = 0; i < OnDetectVfx.Length; i++)
        {
```

```
            OnDetectVfx[i].Stop();
        }

        Animator.SetBool(k_AnimAlertedParameter, false);
    }

    void OnDamaged()
    {
        if (RandomHitSparks.Length > 0)
        {
            int n = Random.Range(0, RandomHitSparks.Length - 1);
            RandomHitSparks[n].Play();
        }

        Animator.SetTrigger(k_AnimOnDamagedParameter);
    }
  }
}
```

Appendix

EnemyTurret.cs

```
using Unity.FPS.Game;
using UnityEngine;

namespace Unity.FPS.AI
{
   [RequireComponent(typeof(EnemyController))]
   public class EnemyTurret : MonoBehaviour
   {
      public enum AIState
      {
         Idle,
         Attack,
      }

      public Transform TurretPivot;
      public Transform TurretAimPoint;
      public Animator Animator;
      public float AimRotationSharpness = 5f;
      public float LookAtRotationSharpness = 2.5f;
      public float DetectionFireDelay = 1f;
      public float AimingTransitionBlendTime = 1f;

      [Tooltip("The random hit damage effects")]
      public ParticleSystem[] RandomHitSparks;

      public ParticleSystem[] OnDetectVfx;
      public AudioClip OnDetectSfx;

      public AIState AiState { get; private set; }

      EnemyController m_EnemyController;
      Health m_Health;
      Quaternion m_RotationWeaponForwardToPivot;
      float m_TimeStartedDetection;
      float m_TimeLostDetection;
      Quaternion m_PreviousPivotAimingRotation;
      Quaternion m_PivotAimingRotation;

      const string k_AnimOnDamagedParameter = "OnDamaged";
      const string k_AnimIsActiveParameter = "IsActive";

      void Start()
      {
         m_Health = GetComponent<Health>();
         DebugUtility.HandleErrorIfNullGetComponent<Health, EnemyTurret>(m_Health, this, gameObject);
         m_Health.OnDamaged += OnDamaged;

         m_EnemyController = GetComponent<EnemyController>();
         DebugUtility.HandleErrorIfNullGetComponent<EnemyController, EnemyTurret>(m_EnemyController, this,
            gameObject);

         m_EnemyController.onDetectedTarget += OnDetectedTarget;
         m_EnemyController.onLostTarget += OnLostTarget;

         // Remember the rotation offset between the pivot's forward and the weapon's forward
         m_RotationWeaponForwardToPivot =
            Quaternion.Inverse(m_EnemyController.GetCurrentWeapon().WeaponMuzzle.rotation) * TurretPivot.rotation;

         // Start with idle
         AiState = AIState.Idle;
```

```
        m_TimeStartedDetection = Mathf.NegativeInfinity;
        m_PreviousPivotAimingRotation = TurretPivot.rotation;
    }

    void Update()
    {
        UpdateCurrentAiState();
    }

    void LateUpdate()
    {
        UpdateTurretAiming();
    }

    void UpdateCurrentAiState()
    {
        // Handle logic
        switch (AiState)
        {
            case AIState.Attack:
                bool mustShoot = Time.time > m_TimeStartedDetection + DetectionFireDelay;
                // Calculate the desired rotation of our turret (aim at target)
                Vector3 directionToTarget =
                    (m_EnemyController.KnownDetectedTarget.transform.position - TurretAimPoint.position).normalized;
                Quaternion offsettedTargetRotation =
                    Quaternion.LookRotation(directionToTarget) * m_RotationWeaponForwardToPivot;
                m_PivotAimingRotation = Quaternion.Slerp(m_PreviousPivotAimingRotation, offsettedTargetRotation,
                    (mustShoot ? AimRotationSharpness : LookAtRotationSharpness) * Time.deltaTime);

                // shoot
                if (mustShoot)
                {
                    Vector3 correctedDirectionToTarget =
                        (m_PivotAimingRotation * Quaternion.Inverse(m_RotationWeaponForwardToPivot)) *
                        Vector3.forward;

                    m_EnemyController.TryAtack(TurretAimPoint.position + correctedDirectionToTarget);
                }

                break;
        }
    }

    void UpdateTurretAiming()
    {
        switch (AiState)
        {
            case AIState.Attack:
                TurretPivot.rotation = m_PivotAimingRotation;
                break;
            default:
                // Use the turret rotation of the animation
                TurretPivot.rotation = Quaternion.Slerp(m_PivotAimingRotation, TurretPivot.rotation,
                    (Time.time - m_TimeLostDetection) / AimingTransitionBlendTime);
                break;
        }

        m_PreviousPivotAimingRotation = TurretPivot.rotation;
    }

    void OnDamaged(float dmg, GameObject source)
    {
        if (RandomHitSparks.Length > 0)
```

```
            {
                int n = Random.Range(0, RandomHitSparks.Length - 1);
                RandomHitSparks[n].Play();
            }

            Animator.SetTrigger(k_AnimOnDamagedParameter);
        }

        void OnDetectedTarget()
        {
            if (AiState == AIState.Idle)
            {
                AiState = AIState.Attack;
            }

            for (int i = 0; i < OnDetectVfx.Length; i++)
            {
                OnDetectVfx[i].Play();
            }

            if (OnDetectSfx)
            {
                AudioUtility.CreateSFX(OnDetectSfx, transform.position, AudioUtility.AudioGroups.EnemyDetection, 1f);
            }

            Animator.SetBool(k_AnimIsActiveParameter, true);
            m_TimeStartedDetection = Time.time;
        }

        void OnLostTarget()
        {
            if (AiState == AIState.Attack)
            {
                AiState = AIState.Idle;
            }

            for (int i = 0; i < OnDetectVfx.Length; i++)
            {
                OnDetectVfx[i].Stop();
            }

            Animator.SetBool(k_AnimIsActiveParameter, false);
            m_TimeLostDetection = Time.time;
        }
    }
}
```